Genuine versus Non-Genuine Atomic Multicast Protocols

Nicolas Schiper[†] Pierre Sutra[‡] Fernando Pedone[†]

[†] University of Lugano
Via Giuseppe Buffi 13
6900 Lugano, Switzerland

[‡] Université Paris VI and INRIA Rocquencourt
104, avenue du président Kennedy
75016 Paris, France

University of Lugano
Faculty of Informatics
Technical Report No. 2009/001
January 2009

Abstract

In this paper, we study atomic multicast, a fundamental abstraction for building fault-tolerant systems. We suppose a system composed of data centers, or groups*, that host many processes connected through high-end local links; a few groups exist, interconnected through high-latency communication links. In this context, a recent paper has shown that no multicast protocol can deliver messages addressed to multiple groups in one inter-group delay and be* genuine*, i.e., to deliver a message m, only the addressees of m are involved in the protocol.*

We first survey and analytically compare existing multicast algorithms to identify latency-optimal multicast algorithms. We then propose a non-genuine multicast protocol that may deliver messages addressed to multiple groups in one inter-group delay. Experimental comparisons against a latency-optimal genuine protocol show that the non-genuine protocol offers better performance in all considered scenarios, except in large and highly loaded systems. To complete our study, we also evaluate a latency-optimal protocol that tolerates disasters, i.e., group crashes.

1 Introduction

Atomic broadcast and multicast are powerful group communication abstractions to build fault-tolerant distributed systems by means of data replication. Informally, they allow messages to be propagated to the group members and ensure agreement on the set of messages delivered and on their delivery order. As opposed to atomic broadcast, atomic multicast allows messages to be addressed to a subset of the members of the system. Multicast can thus be seen as the adequate abstraction for applications in which nodes replicate a subset of the data, i.e., partial replication.

In this paper we consider multicast protocols that span multiple geographical locations. We model the system as a set of *groups*, each one containing *processes* (e.g., data centers hosting local nodes). While a few groups exist, each one can host an arbitrary number of processes. Groups are interconnected through high-latency communication links; processes in a group are connected through high-end local links. The latency and bandwidth of intra- and inter-group links are separated by at least two orders of magnitude, and thus, inter-group links should be used sparingly.

From a problem solvability point of view, atomic multicast can be easily reduced to atomic broadcast: every message is broadcast to all the groups in the system and only delivered by those processes the message is originally addressed to. Such a multicast algorithm is not *genuine* [8] though, since processes not addressed by the message are also involved in the protocol.

Although most multicast algorithms proposed in the literature are genuine (e.g., [7, 6, 13]), it has been shown that genuineness is an expensive property [14]: no genuine atomic multicast algorithm can deliver *global messages*, i.e., messages addressed to multiple groups, in one inter-group message delay,[1] a limitation that is not imposed on non-genuine multicast algorithms. Therefore, when choosing a multicast algorithm, it seems natural to question *the circumstances under which a higher-latency genuine algorithm is more efficient than a non-genuine algorithm, which addresses more processes but delivers messages in one inter-group delay*.

To answer the question, we first survey and analytically compare genuine and non-genuine multicast protocols. We identify inter-group-latency optimal protocols and introduce a novel non-genuine multicast algorithm. This algorithm may deliver global messages in a single inter-group delay (i.e., it is inter-group-latency optimal), and may deliver *local messages*, i.e., messages addressed to a single group, with no inter-group communication. We then experimentally evaluate the three inter-group-latency optimal multicast protocols: two that do not tolerate group crashes, the genuine protocol in [14] and the non-genuine protocol introduced in this paper, and a non-genuine protocol that tolerates group crashes [15], denoted as disaster-tolerant—the only disaster-tolerant genuine multicast we are aware of requires too many inter-group messages to be of practical interest.

We experimentally identify that the genuine protocol may postpone the delivery of messages by as much as two inter-group delays, a phenomenon that we refer to as *convoy effect*. We then revisit the algorithm and propose an optimization to remove the convoy effect for local messages.[2] Although simple, this optimization decreases the delivery latency of local messages by as much as two orders of magnitude.

We then assess the scalability of the multicast protocols by varying the number of groups, the proportion of global messages, and the load, i.e., the frequency at which messages are multicast. The results suggest that the genuineness of multicast is interesting only in large and highly loaded systems, in all the other considered scenarios the non-genuine protocol introduced in this paper outperforms the optimal genuine algorithm. We also show that although the disaster-tolerant multicast protocol is in general more costly than the two other implemented protocols, it matches the performance of the genuine algorithm when there are few groups.

Summing up, this paper makes the following contributions: (a) it surveys multicast algorithms and compare them analytically, (b) it introduces a fast non-genuine multicast algorithm and empirically compares it against other latency-optimal multicast algorithms, and (c) it identifies a convoy effect that slows down the delivery of messages and proposes an optimization to remove this undesirable phenomenon for local messages. Although the phenomenon has been identified in the algorithm in [14], it also happens in other genuine multicast algorithms (e.g., [7, 13]).

The rest of the paper is structured as follows. In Section 2, we introduce the system model and some definitions. Section 3 briefly surveys the existing atomic multicast algorithms and compares them analytically. In Section 4, we present the non-genuine multicast protocol and explain in more detail the protocols of [14, 15]; some implementation issues are discussed in Section 5. The experimental evaluation of the protocols is presented in Section 6 and we conclude the paper in Section 7.

2 System Model and Definitions

2.1 Processes, Links, and Groups

We consider a system $\Pi = \{p_1, ..., p_n\}$ of processes which communicate through message passing and do not

[1] This lower bound is tight since the algorithms in [7, 14] can deliver messages in two inter-group delays.

[2] The convoy effect seem to be unavoidable for global messages.

have access to a shared memory or a global clock. We assume the benign crash-stop failure model: processes may fail by crashing, but do not behave maliciously. A process that never crashes is *correct*; otherwise it is *faulty*. The maximum number of processes that may crash is denoted by f.

The system is asynchronous, i.e., messages may experience arbitrarily large (but finite) delays and there is no bound on relative process speeds. Furthermore, the communication links do not corrupt nor duplicate messages, and are quasi-reliable: if a correct process p sends a message m to a correct process q, then q eventually receives m.[3]

We define $\Gamma = \{g_1, ..., g_m\}$ as the set of process groups in the system. Groups are disjoint, non-empty and satisfy $\bigcup_{g \in \Gamma} g = \Pi$. For each process $p \in \Pi$, $group(p)$ identifies the group p belongs to. Hereafter, we assume that in each group consensus is solvable (the consensus problem is defined below).

2.2 Specifications of Agreement Problems

Consensus Throughout the paper, we assume the existence of a *uniform consensus* abstraction. In the *consensus* problem, processes propose values and must reach agreement on the value decided. Uniform consensus is defined by the primitives propose(v) and decide(v) and satisfies the following properties [9]: (i) *uniform integrity:* if a process decides v, then v was previously proposed by some process, (ii) *termination:* every correct process eventually decides exactly one value, (iii) *uniform agreement:* if a process decides v, then all correct processes eventually decide v.

Reliable Multicast With *reliable multicast*, messages may be addressed to any subset of the groups in Γ. For each message m, $m.dst$ denotes the groups to which the message is reliably multicast. By abuse of notation, we write $p \in m.dst$ instead of $\exists g \in \Gamma : g \in m.dst \wedge p \in g$. Uniform reliable multicast is defined by primitives ***R-MCast(m)*** and ***R-Deliver(m)***, and satisfies the following properties : (i) *uniform integrity:* for any process p and any message m, p R-Delivers m at most once, and only if $p \in m.dst$ and m was previously R-MCast, (ii) *validity:* if a correct process p R-MCasts a message m, then eventually all correct processes $q \in m.dst$ R-Deliver m, (iii) *uniform agreement:* if a process p R-Delivers a message m, then eventually all correct processes $q \in m.dst$ R-Deliver m.

Atomic Multicast *Atomic multicast* is defined by the primitives A-MCast and A-Deliver and satisfies all the properties of reliable multicast as well as *uniform prefix order:* for any two messages m and m' and any two processes p and q such that $\{p, q\} \subseteq m.dst \cap m'.dst$, if p A-Delivers m and q A-Delivers m', then either p A-Delivers m' before m or q A-Delivers m before m'.

In this context, we say that a message m is *local* iff it is addressed to one group only. On the other hand, if m is multicast to multiple groups, it is *global*.

Let $\mathcal{A}$ be an algorithm solving atomic multicast and $\mathcal{R}(\mathcal{A})$ be the set of all admissible runs of $\mathcal{A}$. We define the genuineness of $\mathcal{A}$ as follows [8]:

- *Genuineness*: An algorithm $\mathcal{A}$ solving atomic multicast is *genuine* iff for any run $R \in \mathcal{R}(\mathcal{A})$ and for any process p, in R, if p sends or receives a message then some message m is A-MCast and either p is the process that A-MCasts m or $p \in m.dst$.

3 A Brief Survey of Multicast Algorithms

Although the literature on atomic broadcast algorithms is abundant [5], few atomic multicast protocols exist. We review and compare them analytically. We use two criteria for this comparison: best-case message delivery latency and inter-group message complexity. These metrics are computed by considering a failure-free scenario where a message is A-MCast by some process p to k groups, $k \geq 2$, including $group(p)$. We let δ be the inter-group message delay and assume that the intra-group delay is negligible.

In [8], the authors show the impossibility of solving genuine atomic multicast with unreliable failure detectors when groups are allowed to intersect. Hence, the algorithms cited below consider non-intersecting groups. We first review algorithms that assume that all groups contain at least one correct process, i.e., disaster-vulnerable algorithms, and then algorithms that tolerate group crashes, i.e., disaster-tolerant algorithms.

3.1 Disaster-vulnerable Algorithms

These protocols can be viewed as variations of Skeen's algorithm [3], a multicast algorithm designed for failure-free systems, where messages are associated with timestamps and the message delivery follows the timestamp order.

In [13], the addressees of a message m, i.e., the processes to which m is multicast, exchange the timestamp they assigned to m, and, once they receive this timestamp from a majority of processes of each group, they propose the maximum value received to consensus. Because consensus is run among the addressees of a message and can thus span multiple groups, this algorithm is not well-suited for wide area networks. In the case of a message multicast to multiple groups, the algorithm has a latency of 4δ.

[3] Note that quasi-reliable links can be built on top of lossy links provided that they are *fair*, i.e., not all messages sent are lost [2].

In [6], consensus is run inside groups exclusively. Consider a message m that is multicast to groups $g_1, ..., g_k$. The first destination group of m, g_1, runs consensus to define the final timestamp of m and hands over this message to group g_2. Every subsequent group proceeds similarly up to g_k. To ensure agreement on the message delivery order, before handling other messages, every group waits for a final acknowledgment from group g_k. Hence, the latency degree of this algorithm is $(k+1)\delta$.

In [7], inside each group g, processes implement a logical *clock* that is used to generate timestamps; consensus is used among processes in g to maintain g's clock. Every multicast message m goes through four stages. In the first stage, in every group g addressed by m, processes define a timestamp for m using g's clock. This is g's proposal for m's final timestamp. Groups then exchange their proposals and set m's final timestamp to the maximum among all proposals. In the last two stages, the clock of g is updated to a value bigger than m's final timestamp and m is delivered when its timestamp is the smallest among all messages that are in one of the four stages. In the best case, messages are A-Delivered within 2δ, which is optimal for genuine multicast algorithms [14].

In contrast to [7], [14] allows messages to skip stages, therefore sparing the execution of consensus instances. The best-case latency and the number of inter-group messages sent in [14] are however the same as in [7], as consensus instances are run inside groups. Nevertheless, we observe in Section 6.3.1 that this optimization allows to reduce the *measured* delivery latency under a broad range of loads.

In this paper, we introduce a non-genuine algorithm that is faster than [7] and [14]: it can A-Deliver messages within δ, which is obviously optimal.

3.2 Disaster-tolerant Algorithms

To the best of our knowledge, [15] is the only paper to address the problem of group crashes. Two protocols are presented. The first one is genuine and tolerates an arbitrary number of failures but requires perfect failure detection and has a latency of 6δ, it is therefore not suited for wide area networks. The second algorithm is not genuine but only requires perfect failure detection inside each group. It can deliver messages within only 2δ.[4] It however requires a two-third majority of correct processes, i.e., $f < n/3$. As a corollary of [11], this protocol is optimal.[5]

[4] This algorithm can also tolerate unreliable failure detection, but at the cost of a weaker liveness guarantee.

[5] To tolerate group crashes and deliver global messages in 2δ, a non-genuine algorithm such as Fast Paxos [10] could also be used. However, this algorithm achieves a latency of 2δ only when messages are spontaneously ordered, an assumption [15] does not need. Moreover, in contrast to Fast Paxos, [15] may deliver local messages with no inter-group communication.

3.3 Analytical Comparison

Figure 1 provides a comparison of the presented algorithms. To compute the inter-group message complexity, we assume that there are n processes in the system and that every group is composed of d processes. The third and the fifth column respectively indicate whether the protocol tolerates group crashes and whether it is latency-optimal.

Algorithm	genuine?	disaster tolerant?	latency	inter-group msgs.	latency optimal?
[6]	yes	no	$(k+1)\delta$	$O(kd^2)$	no
[13]	yes	no	4δ	$O(k^2d^2)$	no
[7, 14]	yes	no	2δ	$O(k^2d^2)$	yes
this paper	no	no	δ	$O(n^2)$	yes
[15]	yes	yes	6δ	$O(k^3d^3)$	no
[15]	no	yes	2δ	$O(n^2)$	yes

Figure 1. Comparison of the algorithms (d : nb. of processes per group, k : nb. of destination groups)

4 Optimal Multicast Algorithms

To experimentally evaluate and compare genuine and non-genuine atomic multicast protocols, we picked the algorithms achieving optimal latency in each category.[6] We begin by presenting protocols that do are disaster-vulnerable, namely the genuine algorithm of [14], denoted as $\mathcal{A}_{ge}$, and the non-genuine algorithm introduced in this paper, denoted as $\mathcal{A}_{ng}$. This latter algorithm delivers messages faster than $\mathcal{A}_{ge}$: $\mathcal{A}_{ng}$ has delivery latency of δ units of time. We then present the non-genuine and disaster-tolerant algorithm of [15], $\mathcal{A}_{dt}$.

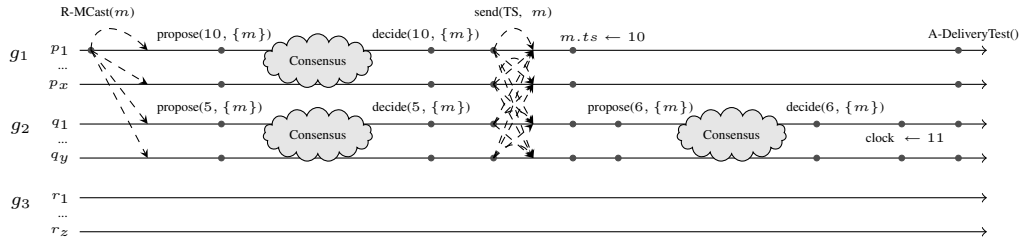
4.1 Genuine Multicast ($\mathcal{A}_{ge}$)

In Algorithm $\mathcal{A}_{ge}$, to multicast a message m, m is first reliably multicast to its addressees and then assigned a global unique timestamp. To ensure agreement on the message delivery order, two properties are ensured: (1) processes agree on the message timestamps and (2) after a process p A-Delivers a message with timestamp ts, p does not A-Deliver a message with a smaller timestamp than ts. To satisfy these two properties, inside each group g, processes implement a logical *clock* that is used to generate timestamps and use consensus to update it.

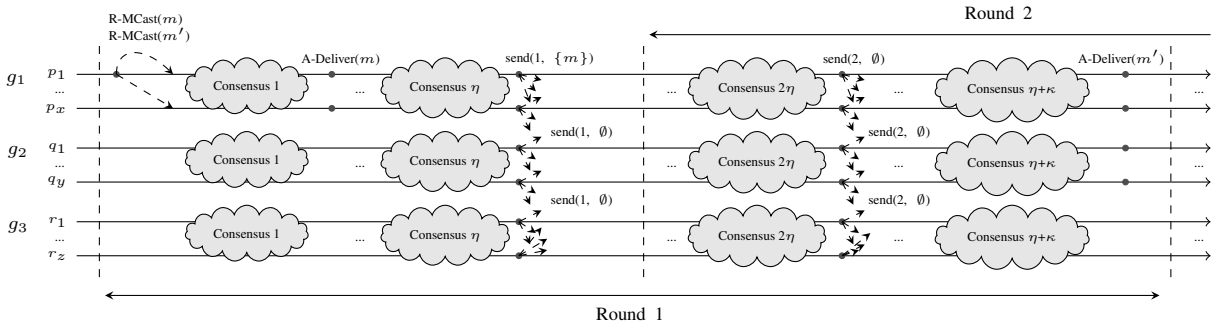
In more detail, every multicast message m goes trough the following four stages:

- *Stage* s_0: In every group $g \in m.dst$, processes define a timestamp for m using g's clock. This is g's proposal for m's final timestamp.

[6] Another criterion for choosing the algorithms could have been inter-group message complexity. However, existing algorithms that exhibit a lower message complexity than the ones we picked have a prohibitive latency.



(a) Algorithm $\mathcal{A}_{ge}$ when a message m is A-MCast to groups g_1 and g_2.



(b) Algorithm $\mathcal{A}_{ng}$ when messages m and m' are A-MCast from p_1 and are respectively addressed to g_1 and $\{g_2, g_3\}$.

Figure 2. Algorithms $\mathcal{A}_{ge}$ and $\mathcal{A}_{ng}$ in the failure-free case.

- *Stage* s_1: Groups in $m.dst$ exchange their proposals for m's timestamp and set m's final timestamp to the maximum timestamp among all proposals.
- *Stage* s_2: Every group in $m.dst$ sets its clock to a value greater than the final timestamp of m.
- *Stage* s_3: Message m is A-Delivered when its timestamp is the smallest among all messages that are in one of the four stages and not yet A-Delivered.

Timestamps can be implemented in different ways. For example, each group g addressed by message m can define g's timestamp by using the consensus instance number that decides on m (stage s_1). Moreover, a consensus instance may decide on multiple messages, possibly in different stages. Since every consensus instance i may decide on messages in stage s_2, after deciding in i, g's clock is set to one plus the biggest message timestamp that i decided on.

In contrast to [7], not all messages go trough all four stages in $\mathcal{A}_{ge}$. Messages that are multicast to one group only can *jump* from stage s_0 to stage s_3. Moreover, even if a message m is multicast to more than one group, on processes belonging to the group that proposed the largest timestamp, i.e., m's final timestamp, m skips stage s_2.

Figure 2 (a) illustrates a failure-free run of the algorithm in which, at the beginning of the run, groups g_1 and g_2 have their clock equal to 10 and 5 respectively.

4.2 Non-Genuine Multicast ($\mathcal{A}_{ng}$)

Algorithm $\mathcal{A}_{ng}$ works as follows (see Figure 2 (b)). To A-MCast a message m, a process p R-MCasts m to p's group. In parallel, processes execute an *unbounded* sequence of rounds. At the end of each round, processes A-Deliver a set of messages according to some deterministic order. To ensure agreement on the messages A-Delivered in round r, processes proceed in two steps: In the first step, inside each group g, processes use consensus to define g's bundle of messages. In the second step, groups exchange their message bundles. The set of message A-Delivered by some process p at the end of round r is the union of all bundles, restricted to messages addressed to p.

Notice that in the current protocol, *local messages*, i.e., messages multicast to a single group, are delivered only after receiving the groups' bundle of messages. This is however unnecessary: local messages can be delivered directly after consensus since they are addressed to a single group, and thus before receiving the groups' message bundles. Hence, these messages do not bear the cost of a single inter-group delay unless: (a) they are multicast from a group different than their destination group or (b) they are multicast while the groups' bundle of messages are being exchanged. Obviously, nothing can be done to avoid case (a) from happening. However, we can make case (b) unlikely to happen by executing multiple consensus instances per round. The number of consensus instances per round is denoted by parameter κ.

Although the two above optimizations decrease the average delivery latency of local messages, the delivery latency of *global messages*, i.e., messages that are not local, can be increased by as many as κ - 1 consensus instances (this is

because each group's bundle of messages is sent every κ consensus instance). Hence, to reduce the delivery latency of global messages, we allow rounds to overlap. That is, we start the next round before receiving the groups' bundle of messages of the current round. In other words, we execute consensus instances while the groups' bundle of messages are being exchanged. In our implementation, message bundles are exchanged after every η consensus instances.

To ensure agreement on the relative delivery order of local and global messages, it is necessary that processes inside the same group agree on when global messages of a given round are delivered, i.e., after which consensus instance. To summarize, processes send the message bundle of some round r after consensus instance $r \cdot \eta$ and A-Deliver messages of round r after instance $r \cdot \eta + \kappa$. Section 6.2 explores the influence of these parameters on the protocol.

4.3 Disaster-tolerant Multicast ($\mathcal{A}_{dt}$)

Algorithm $\mathcal{A}_{dt}$ is similar to Algorithm $\mathcal{A}_{ng}$. To cope with group crashes, the exchange of message bundles is handled differently however. Indeed, in case some group g crashes, $\mathcal{A}_{ng}$ does not ensure liveness as there will be some round r after which no process receives the message bundles from g. To circumvent this problem we proceed in two steps: (a) we allow processes to stop waiting for g's message bundle, and (b) we let processes agree on the set of message bundles to consider for each round.

To implement (a), processes maintain a common *view* of the groups that are trusted to be alive, i.e., groups that contain at least one alive process. Processes then wait for the message bundles from the groups currently in the view. A group g may be erroneously removed from the view, if it was mistakenly suspected of having crashed. Therefore, to ensure that message m multicast by a correct process will be delivered by all correct addressees of m, we allow members of g to add their group back to the view. To achieve (b), processes agree on the sequence of views and the set of message bundles between each view change. For this purpose, we use a generic broadcast abstraction [12] to propagate message bundles and view change messages, i.e., messages to add or remove groups.[7] Since message bundles can be delivered in different orders at different processes, provided that they are delivered between the same two view change messages, we define the message conflict relation as follows: view change messages conflict with all messages and message bundles only conflict with view change messages. As view change messages are not expected to be broadcast often, such a conflict relation definition allows for fast message delivery (i.e., within two inter-group delays), which is optimal [11].

[7] Informally, generic broadcast ensures the same properties as atomic multicast except that messages are always addressed to all groups and only *conflicting* messages are totally ordered. That is, the uniform prefix order property of Section 2.2 is only ensured for messages that conflict.

5 Implementation Issues

We here present some key points of the algorithms' implementation. The three algorithms of Section 4 were implemented in Java and use a Paxos library as the consensus protocol [4]; all communications are based on TCP.

Inter-group communication represents a major source of overhead and should thus be used sparingly. In our implementation, these communications are thus handled by a dedicated layer. As we explain below, this layer achieves some optimizations in order to reduce the number of inter-group messages sent.

Message Batching. Inside each group g, a special process is elected as leader [16]. Members of a group use their leader to forward messages to the remote groups' leaders. When a leader receives a message m, it dispatches m to the members of its group.

Message Filtering. In each one of the presented algorithms, inter-group communication originating from processes of the same group g presents some redundancy. In the non-genuine Algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$, at the end of a round r, members of g send the same message bundle. Moreover, in the genuine Algorithm $\mathcal{A}_{ge}$, members of g send the same timestamp proposal for some message m. To avoid this redundancy, only the groups' leaders propagate these messages. More precisely, message bundles of Algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$, and the timestamp proposals of Algorithm $\mathcal{A}_{ge}$ are only sent by the groups' leaders. Messages sent by non-leader processes are discarded by the inter-group communication layer.

In the case of a leader failure, these optimizations may lead to the loss of some messages, these messages will thus have to be resent.

6 Experimental evaluation

In this section, we evaluate experimentally the performance of optimal multicast protocols. We start by describing the system parameters and the benchmark used to assess the protocols. We then explore the influence of κ and η on $\mathcal{A}_{ng}$; compare the genuine and non-genuine protocols by varying the load imposed on the system, the number of groups, and the proportion of global messages; and measure the overhead of tolerating disasters.

6.1 Experimental Settings

The system. The experiments were conducted in a cluster of 24 nodes connected with a gigabit switch. Each node is equipped with two dual-core AMD Opteron 2 Ghz, 4GB of RAM, and runs Ubuntu Linux 4. In all experiments,

each group consists of 3 nodes; the number of groups varies from 4 to 8. The bandwidth and message delay of our local network, measured using netperf and ping, are about 940 Mbps and 0.05 ms respectively. To emulate inter-group delays with higher latency and lower bandwidth, we used the Linux traffic shaping tools.

We emulated two network setups. In setup 1, the message delay between any two groups follows a normal distribution with a mean of 100 ms and a standard deviation of 5 ms, and each group is connected to the other groups via a 125 KBps (1 Mbps) full-duplex link. In setup 2, the message delay between any two groups follows a normal distribution with a mean of 20 ms and a standard deviation of 1 ms, and each group is connected to the other groups via a 1.25MBps (10 Mbps) full-duplex link. Due to space constraints, we only report the results using setup 1 and briefly comment on the behavior of the algorithms in setup 2.

The benchmark. The communication pattern of our benchmark was modeled after TPC-C, an industry standard benchmark for on-line transaction processing (OLTP) [1]. TPC-C represents a generic wholesale supplier workload and is composed of five predefined transaction types. Two out of these five types may access multiple warehouses; the other three types access one warehouse only. We assume that each group hosts one warehouse. Hence, the warehouses involved in the execution of a transaction define to which group the transaction is multicast. Each multicast message also contain the transaction's parameters and on average, a message contains 80 bytes of payload.

In TPC-C, about 10% of transactions involve multiple warehouses. Thus, roughly 10% of messages are global. To assess the scalability of our protocols, we parameterize the benchmark to control the proportion p of global messages. In the experiments, we report measurements for $p = 0.1$ (i.e., original TPC-C) and $p = 0.5$. The vast majority of global messages involve two groups. Note that in our benchmark, transactions are not executed; TPC-C is only used to determine the communication pattern.

Each node of the system contains an equal number of clients executing the benchmark in a closed loop: each client multicasts a message and waits for its delivery before multicasting another message. Hence, all messages always address the sender's group; global messages also address other groups. The number of clients per node varies between 1 and 160 and in each experiment, at least one hundred thousand messages are multicast.

For all the experiments, we report either the average message delivery latency (in milliseconds) or the average inter-group bandwidth (in kilo bytes per second) as a function of the throughput, i.e., the number of messages A-Delivered per minute. We computed 95% confidence intervals for the A-delivery latency but we do not report them here as they were always smaller than 2% of the average latency. The throughput was increased by adding an equal number of clients to each node of the system.

6.2 The influence of κ and η on $\mathcal{A}_{ng}$

Setting parameters κ, the number of consensus instances per round, and η, the number of consensus instances between two consecutive message bundle exchanges, is no easy task. In principle, the optimal value of κ should be so that groups execute as many local consensus instances as they can while message bundles are being exchanged, but not too many. If we let Δ_{max} and δ_{cons} respectively denote the maximum inter-group delay and the consensus latency, κ should be set to $\lfloor \Delta_{max}/\delta_{cons} \rfloor$. Setting parameter η is however less trivial: setting it low potentially decreases the average global message latency but may also saturate the inter-group network.

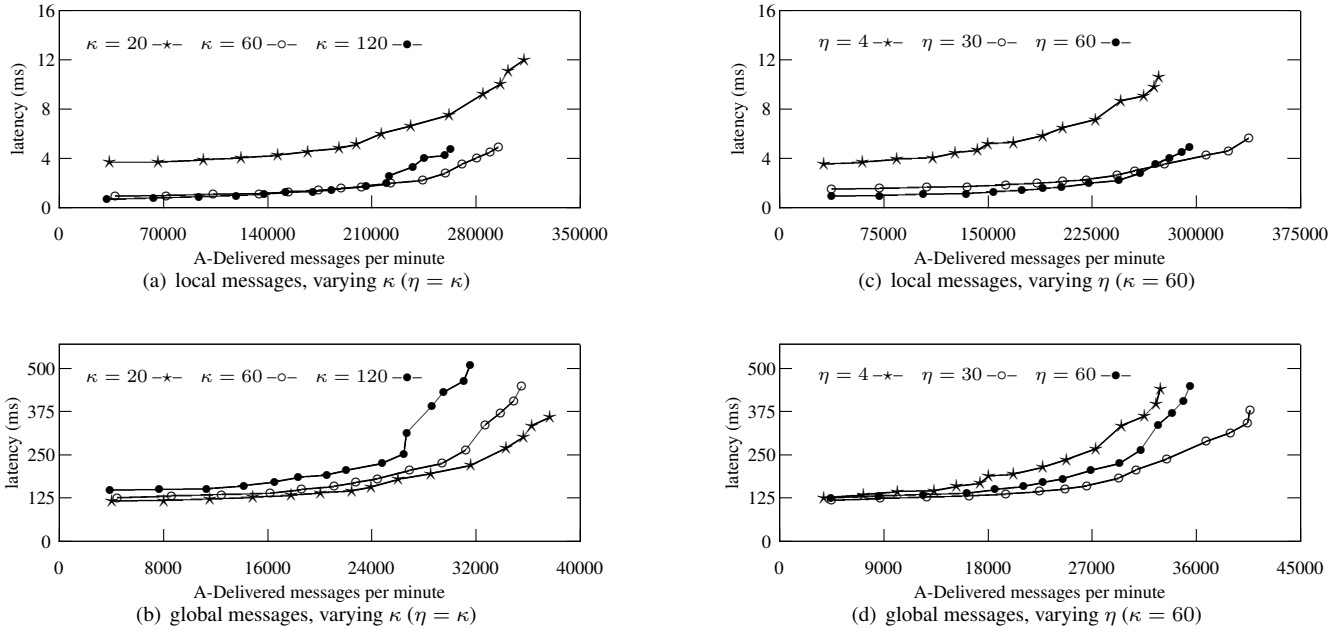
In Figures 3(a) and 3(b), we first explore the influence of κ on the protocol in a system with four groups when rounds do not overlap (i.e., $\eta = \kappa$). We first consider local messages, and note that setting κ too low (i.e., $\kappa = 20$ in our experiments) or too high ($\kappa = 120$) respectively increases the latency or harms the scalability of the protocol: as the number of global messages sent at the end of each round increased, the inter-group communication became the bottleneck since the traffic was too bursty. In our settings, $\kappa = 60$ gave the best results for local messages (see Figure 3(a)).

For global messages, setting κ to 60 gives worse performance than setting it to 20 (see Figure 3(b)). We address this problem by tuning parameter η, as illustrated in Figures 3(c) and 3(d). While setting η too low ($\eta = 4$) worsens the latency and the scalability of the protocol, setting it too high ($\eta = 60$) harms its scalability. When $\eta = 30$, the local message latency is similar to the one when $\kappa = \eta = 60$ (Figure 3(a)), while almost matching the global message latency of $\kappa = \eta = 20$ (Figure 3(b)). Moreover, with respect to Figures 3(a) and 3(b), the scalability of the protocol is improved for both local and global messages.

To further reduce the global message latency, we tried other values for η. However, we did not find a value that gave better performance than $\eta = 30$ nor did we reach the theoretical optimum latency of one δ, i.e., 100 ms. We observed that this was mainly because groups do not start rounds exactly at the same time; consequently, some groups had to wait more than 100 milliseconds to receive all message bundles. Therefore, in all experiments that follow we used $\kappa = 60$ and $\eta = 30$.

6.3 Genuine vs. Non-Genuine Multicast

We now compare the non-genuine algorithm $\mathcal{A}_{ng}$ to its genuine counterpart $\mathcal{A}_{ge}$ in a system with four groups (see



(a) local messages, varying κ ($\eta = \kappa$)
(c) local messages, varying η ($\kappa = 60$)
(b) global messages, varying κ ($\eta = \kappa$)
(d) global messages, varying η ($\kappa = 60$)

Figure 3. The influence of κ and η on $\mathcal{A}_{ng}$ in a system with four groups.

Figures 4(a) and 4(b)). From the results, we observe that the non-genuine algorithm delivers local and global messages faster than its genuine counterpart, except under very high loads (Figure 4(b)). For global messages, this is a direct consequence of the fact that the best-case delivery latency of $\mathcal{A}_{ng}$ is δ but it is of 2δ for $\mathcal{A}_{ge}$. In fact, under low loads, the latency of $\mathcal{A}_{ge}$ was only a few milliseconds higher than 200 milliseconds. For local messages it is however less obvious why $\mathcal{A}_{ge}$ takes between 65 and 240 milliseconds to deliver local messages while $\mathcal{A}_{ng}$ only takes a few milliseconds (Figure 4(a)). In fact, this phenomenon is due to global messages that slow down the delivery of local messages as we now explain.

Consider a global message m_1 and a local message m_2 that are addressed to groups $\{g_1, g_2\}$ and g_1 respectively. Processes in g_1 R-Deliver m_1 and define their proposal timestamp for m_1 with consensus instance k_1. Shortly after, members of g_1 R-Deliver m_2 and decide on m_2 in consensus instance k_2, such that $k_2 > k_1$. Message m_2 cannot be delivered at this point since m_1 has a smaller timestamp than m_2 and m_1 has not been delivered yet. To deliver the local message m_2, members of g_1 must wait to receive g_2's timestamp proposal for m_1, which may take up to 2δ, i.e., 200 milliseconds, if m_1 was A-MCast from within g_1.

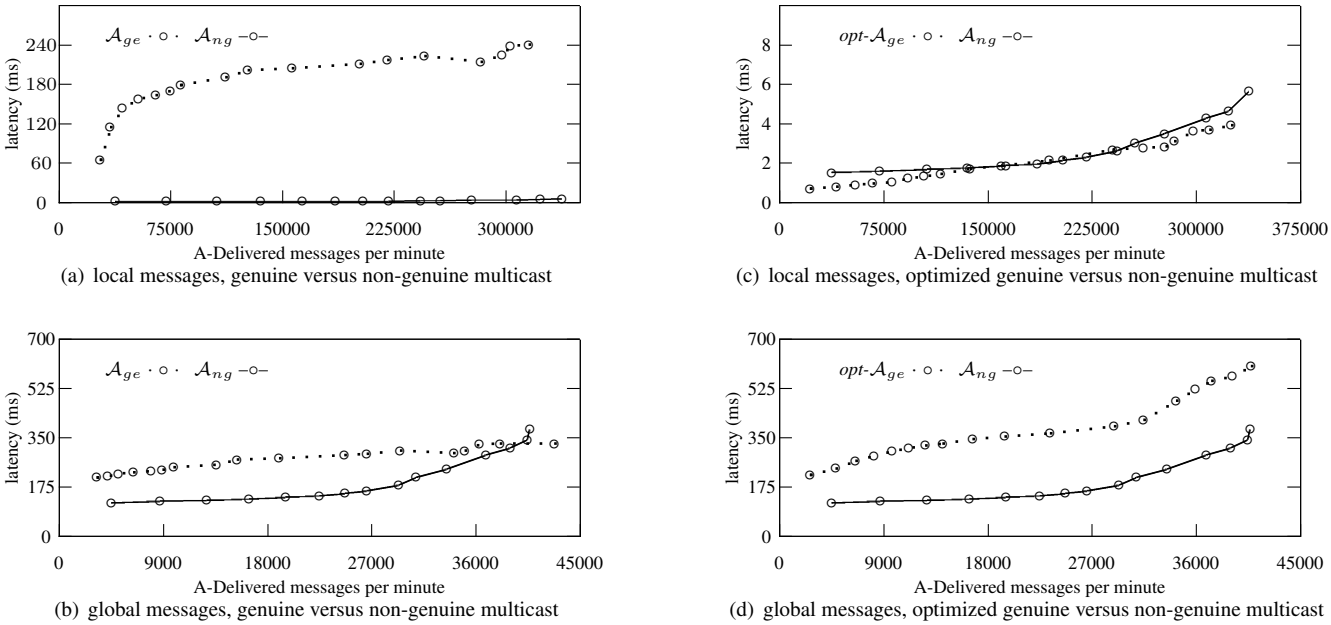
In the scenario described above, m_2 is experiencing what we denote as the *convoy effect*, i.e., m_1 gets a smaller temporary timestamp than m_2 right before m_2 could have been delivered and postpones m_2's delivery. Naturally, the convoy effect can also happen to global messages: in the scenario described above, if we let m_2 be a global message and consider that processes in g_1 decide on m_2 for the second time in consensus instance k_2, m_2 could be delayed by 2δ and thus be delivered within as much as 400 milliseconds. Below, we explain how we modified $\mathcal{A}_{ge}$ to remove the convoy effect for local messages. For global messages however, the convoy effect seems to be unavoidable if agreement on their delivery order must be ensured.

6.3.1 Optimizing the Genuine Algorithm

To deliver local messages faster, we handle global and local messages differently. Local messages are not assigned timestamps anymore and are A-Delivered directly after consensus. More precisely, when some process wishes to A-MCast a local message m to a group g, m is reliably multicast to g, as in $\mathcal{A}_{ge}$. In each group of the system, we run consensus instances to ensure agreement on the delivery order of local messages, as well as to assign timestamps to global messages as explained in Section 4.1. As soon as a consensus instance in g decides on m, members of g A-Deliver m.

To agree on the delivery order of global and local messages, global messages must be A-Delivered after the same consensus instance on members of the same group. To ensure this property, all global messages m must go through the four stages of Section 4.1, even in the group that proposed the highest timestamp for m.

To understand why this is necessary, consider a global message m_1 and a local message m_2 that are respectively addressed to groups $\{g_1, g_2\}$ and g_1. Group g_1 is the group that assigned the highest timestamp to m_1. If we allow m_1



(a) local messages, genuine versus non-genuine multicast

(c) local messages, optimized genuine versus non-genuine multicast

(b) global messages, genuine versus non-genuine multicast

(d) global messages, optimized genuine versus non-genuine multicast

Figure 4. Genuine versus Non-Genuine Multicast in a system with four groups.

to skip stage s_2 in g_1, two members p and q of g_1 may A-Deliver m_1 and m_2 in different orders. For example, assume processes p and q define m_1's proposal timestamp in a consensus instance k_1. Then, p receives g_2's timestamp for m_1, A-Delivers m_1, decides on m_2 in a consensus instance k_2, and A-Delivers m_2. However, q first decides in consensus instance k_2, delivers m_2, receives g_2's timestamp proposal for m_1, and delivers m_1.

Figures 4(c) and 4(d) compare the performance of $\mathcal{A}_{ng}$ to the optimized version of $\mathcal{A}_{ge}$, hereafter *opt-*$\mathcal{A}_{ge}$, in a system with four groups. Algorithm *opt-*$\mathcal{A}_{ge}$ delivers local messages as fast as $\mathcal{A}_{ng}$ (see Figure 4(c)) but slows down the delivery of global messages (see Figures 4(b) and 4(d)). This phenomenon has two causes. First, all global messages now go through the four stages, thus, an increased number of consensus instances must be run for the same throughput. Second, as an effect of the first cause, global messages have a larger window of vulnerability to the convoy effect. Hence, *opt-*$\mathcal{A}_{ge}$ is interesting only when the decrease in local message latency it offers matters more than the increase in global message latency.

Finally, note that the above observations underline the benefit of allowing any global message m to skip stage s_2 in the group that proposed m's largest timestamp: under a high load of global messages $\mathcal{A}_{ge}$ would deliver global messages faster than its non-optimized counterpart [7].

6.3.2 Scalability

We now compare $\mathcal{A}_{ng}$ to *opt-*$\mathcal{A}_{ge}$ when the number of groups increases using two mixes of global and local messages. We first set the proportion of global messages to 10% and run the algorithms in a system with four and eight groups. Figures 5(a) and 5(b) respectively report the average outgoing inter-group traffic per group and the average A-Delivery latency, both as a function of the throughput. For brevity we report the overall average delivery latency, without differentiating between local and global messages.

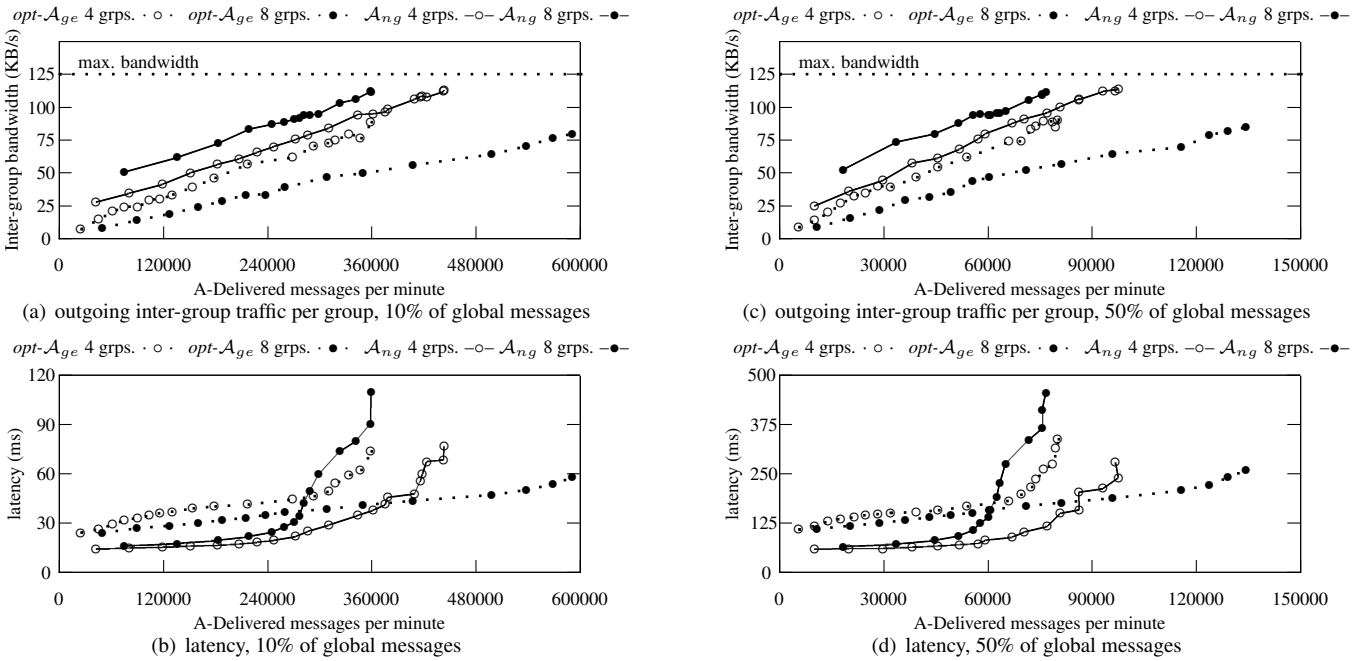
Although $\mathcal{A}_{ng}$ exhibits a better latency than *opt-*$\mathcal{A}_{ge}$ with 4 groups, $\mathcal{A}_{ng}$ does not scale as well as *opt-*$\mathcal{A}_{ge}$ with eight groups (Figure 5(b)). This is a consequence of $\mathcal{A}_{ng}$'s higher demand on throughput: with eight groups the algorithm requires as much as 111 KBps of average inter-group bandwidth, a value close to the maximum available capacity of 125 KBps (Figure 5(a)).

In Figures 5(c) and 5(d), we observe that when half of the messages are global, the two algorithms compare similarly as above but do not scale as well.

As a final remark, we note that in contrast to $\mathcal{A}_{ng}$, *opt-*$\mathcal{A}_{ge}$ delivers messages faster and supports more load with eight groups than with four (Figures 5(b) and 5(d)). Indeed, increasing the number of groups decreases the load that each group must handle as, in our benchmark, the vast majority of global messages are addressed to two groups. This effect can be seen in Figures 5(a) and 5(c), where each group needs less inter-group bandwidth with eight groups.

6.3.3 Summary

Figure 6 provides a qualitative comparison between the genuine and non-genuine algorithms. We consider four scenar-



(a) outgoing inter-group traffic per group, 10% of global messages

(c) outgoing inter-group traffic per group, 50% of global messages

(b) latency, 10% of global messages

(d) latency, 50% of global messages

Figure 5. The scalability of $\mathcal{A}_{ng}$ and *opt-*$\mathcal{A}_{ge}$.

ios generated by all combinations of the two following parameters: the load (high or low) and the number of groups (many or few); the proportion of global messages is not taken into account as it has no influence on the comparison. We note that $\mathcal{A}_{ng}$ is the winner except when the load is high and there are many groups.

We also carried out the same comparison in network setup 2, i.e., a network where the message delay between any two groups follows a normal distribution with a mean of 20 ms and a standard deviation of 1 ms, and each group is connected to the other groups via a 1.25MBps (10 Mbps) full-duplex link. Due to space constraints we only briefly comment on the obtained results. With 4 groups, *opt-*$\mathcal{A}_{ge}$ and $\mathcal{A}_{ng}$ compare similarly as in setup 1. Due to the lower inter-group latency the performance of *opt-*$\mathcal{A}_{ge}$ becomes closer to the one of $\mathcal{A}_{ng}$ however. With eight groups, the non-genuine protocol scales almost as well as the genuine algorithm thanks to the extra available inter-group bandwidth.

6.4 The Cost of Tolerating Disasters

To evaluate the overhead of tolerating disasters, we compare $\mathcal{A}_{dt}$ to the overall best-performing disaster-vulnerable algorithm $\mathcal{A}_{ng}$. With $\mathcal{A}_{dt}$ we set κ and η to 120 and 60 respectively; with $\mathcal{A}_{ng}$, the default values are used, $\kappa = 60$ and $\eta = 30$.

In Figures 7(b) and 7(d), we observe that with four groups, $\mathcal{A}_{dt}$ roughly needs twice as much time as $\mathcal{A}_{ng}$ to deliver messages. This is expected: local messages take about the same time to be delivered with the two algorithms;

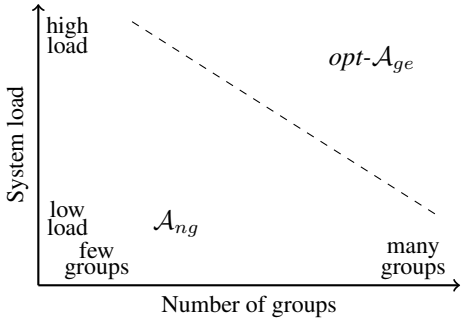


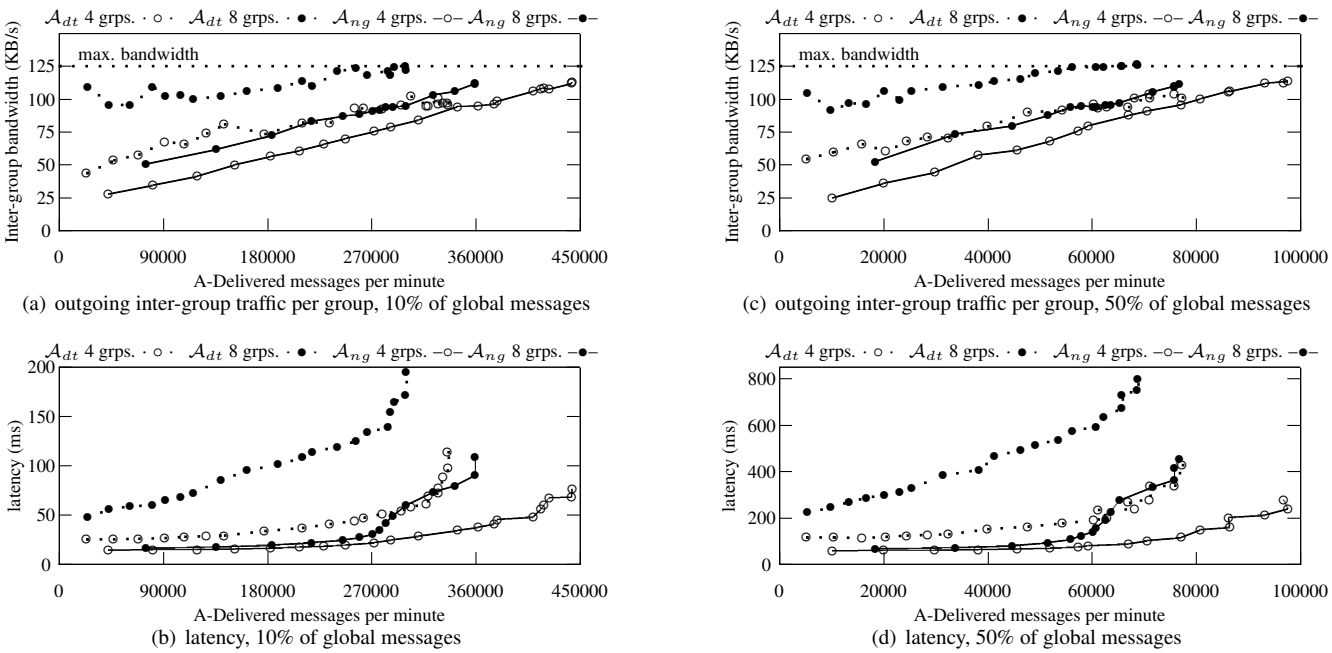
Figure 6. Comparing *opt-*$\mathcal{A}_{ge}$ to $\mathcal{A}_{ng}$.

global messages roughly need and additional 100 milliseconds to be delivered with $\mathcal{A}_{dt}$. Interestingly, $\mathcal{A}_{dt}$ matches the performance of $\mathcal{A}_{ge}$ in a system with four groups (Figures 5(b), 7(b), 5(d), and 7(d)).

With eight groups, $\mathcal{A}_{dt}$ utilizes the entire inter-group bandwidth under almost every considered load (Figures 7(a) and 7(c)). The latency and scalability of the disaster-tolerant algorithm thus become much worse than $\mathcal{A}_{ng}$'s.

7 Conclusion

In this paper, we surveyed and analytically compared existing multicast algorithms. We then proposed a non-genuine multicast algorithm that improves the latency of known protocols by one inter-group message delay. Experimental comparisons against the latency-optimal genuine algorithm in [14] show that the non-genuine protocol offers



(a) outgoing inter-group traffic per group, 10% of global messages

(b) latency, 10% of global messages

(c) outgoing inter-group traffic per group, 50% of global messages

(d) latency, 50% of global messages

Figure 7. The cost of tolerating disasters.

better performance except in large and highly loaded systems. To complete our study, we also evaluated a protocol that tolerates disasters [15]. Although it does not offer the same level of performance as the non-genuine algorithm introduced in this paper, [15] matches the performance of [14] when there are few groups.

As future work, we intend to combine the low latency of the non-genuine algorithm with the extra scalability of [14] in a *hybrid* algorithm: under low loads, the non-genuine algorithm is used; when the load becomes too important, and thus the latency increases drastically, processes resort to the genuine protocol.

References

[1] Transaction processing performance council (tpc) - benchmark c. http://www.tpc.org/tpcc/.

[2] M. K. Aguilera, W. Chen, and S. Toueg. On quiescent reliable communication. *SIAM J. Comput.*, 29(6):2040–2073, 2000.

[3] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.

[4] L. Camargos. http://sourceforge.net/projects/daisylib/.

[5] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[6] C. Delporte-Gallet and H. Fauconnier. Fault-tolerant genuine atomic multicast to multiple groups. In *Proceedings of OPODIS'00*, pages 107–122. Suger, Saint-Denis, rue Catulienne, France, 2000.

[7] U. Fritzke, P. Ingels, A. Mostéfaoui, and M. Raynal. Fault-tolerant total order multicast to asynchronous groups. In *Proceedings of SRDS'98*, pages 578–585. IEEE Computer Society, 1998.

[8] R. Guerraoui and A. Schiper. Genuine atomic multicast in asynchronous distributed systems. *Theoretical Comput. Sci.*, 254(1-2):297–316, 2001.

[9] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

[10] L. Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[11] L. Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, 2006.

[12] F. Pedone and A. Schiper. Handling message semantics with generic broadcast protocols. *Distributed Computing*, 15(2):97–107, 2002.

[13] L. Rodrigues, R. Guerraoui, and A. Schiper. Scalable atomic multicast. In *Proceedings of IC3N'98*, pages 840–847. IEEE, 1998.

[14] N. Schiper and F. Pedone. On the inherent cost of atomic broadcast and multicast in wide area networks. In *Proceedings of ICDCN'08*, pages 147–157. Springer, 2008.

[15] N. Schiper and F. Pedone. Solving atomic multicast when groups crash. Technical Report 2008/002, University of Lugano, 2008. To appear in Proceedings of OPODIS'08.

[16] N. Schiper and S. Toueg. A robust and lightweight stable leader election service for dynamic systems. In *Proceedings of DSN'08*, pages 207–216. IEEE Computer Society, 2008.